Kodningsstandard i C++ för the invisible pixels

Benjamin Ingberg

18 februari 2013

Innehåll

Sammanfattning

Detta dokument beskriver kodningsstandarden vi ska hålla oss till inom vår projektgrupp för att få en kodbas som är lättare att underhålla och förstå. Det är både en blandning utav regler och lösa riktlinjer och tips för att göra koden mer läsbar.

Ingen utav reglerna är skrivna i sten, men generellt sett ska man inte bryta en regel om man inte har en riktigt bra motivation, men om man förenklar koden genom att bryta mot regeln så går det bra att bryta mot den.

Detta dokument är löst baserat på google standarden för C++.

1 Kodstil

Generellt så ska kod följa följande mönster.

```
template<typename D>
class MinKlass {

    D minPrivataVariabel;
public:
    MinKlass(D in = D()) : minPrivataVariabel(in) { }

    /// <summary> Skriver ut resultatet. </summary>
    /// <param id="out"> Streamobjeket som används. </param>
    /// <return> Samma streamobjekt för att kunna kedja.
    /// </return>
    ostream & print(ostream & out) {
        out << minPrivataVariabel;
    }

    /// <summary> Kontrollerar om talet är positivt.
    /// </summary>
    /// <return> Sant om talet är positivt. </return>
    bool isPositive() {
        if(D > 0) // ensamma satser utan klammrar är ok
            return true;

        // om någon av satserna kräver klammrar så måste
        // alla satser ha klammrar
        if(D < 0) {
            return false;
        }
        else if(D == 0) {
            cout << "Är noll positivt?" << endl;
            throw logic_error("hur var det nu");
        }
    }
};
```

Detta är alltså tabbar motsvarande 4-blankstegs indentering. Detta är, lyckligtvis, standard för visual studio men om ni av någon anledning inte har denna indentering gå in i TOOLS->Options->Text Editor->C/C++->Tabs. Där klickar ni i tab size 4, indent size 4 och Keep Tabs.

2 Headerfiler

Header filer definierar funktionalitet i C++, de innehåller deklarationer utav allt som en källkodsfil kan göra som ej är strikt lokal funktionalitet.

Generellt sett ska varje .cpp fil ha en motsvarande .hpp fil som deklarerar allting som behövs av .cpp filen.

2.1 Header guards

Alla header filer ska innehålla header guards vare sig de behövs eller ej. Header guards ser ut på följande sätt:

```
#ifndef _NAMN_HPP
#define _NAMN_HPP
        // resten av filen
#endif
```

Utan header guards kan samma header fil inkluderas flera gånger i ett dokument vilket leder till att man får kompileringsfel.

2.2 Header filer för C

Om en del av koden är skriven i C istället för C++ så ska header filen heta .h och källkodsfilen heta .c, headerfilen behöver även ett makro för att länkning ska fungera.

```
#ifdef __cplusplus
extern "C" {
#endif
        // resten av filen
#ifdef __cplusplus
}
#endif
```

Detta makro stoppas under header guarden.

2.3 Fördeklarering

Ingen fördeklarering, om din kod är beroende av kod från en annan header fil så måste du inkludera denna andra header fil.

2.4 Inline

Använd inline som nyckelord innan en funktionsdeklaration om (och endast om) både definitionen och deklarationen görs i headerfilen (gäller ej klasser, klassmetoder eller template funktioner).

Annars kan det uppstå skumma kompileringsfel. Skriv generellt sett inte definitionen i header filen.

3 Scope-regler

Ett scope är ett område under vilket man kan komma åt variabler, funktioner och klasser som tidigare har deklarerats. Se exempel:

```
// globalscope
namespace namn {
        // namespace-scope
        void funktion() {
                // funktionsscope del 1
                while(true) {
                        // whilescope
                        {
                                // lokalscope
                        }
                }
                // funktionsscope del 2
        }
}
```

Man kan alltid komma åt det som definierats tidigare i ett ovanstående scope, i ovanstånde exempel kan man i lokalscope komma åt allting som definierats innan, men i funktionsscope 2 så kommer man ej åt det som definierats i whilescope och lokalscope.

På svenska så uttalas scope som skåp.

3.1 Lokala variabler

Lokala variabler ska deklareras så sent som möjligt. Dvs i det djupaste scopet som där man fortfarande kommer åt det man vill ha.

4 Namespaces

Namespaces ska alltid användas. Inuti källkodsfiler ska anonyma namespaces användas (ett namespace utan namn) för funktionalitet som är onödig utanför filen.

I header filer måste namespaces alltid specifieras och inga using statements får användas (förutom i lokalt scope).